MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

AI Memo No. 378 September 1976

Arithmetic Shifting Considered Harmful

by

Guy L. Steele Jr. *

Abstract:

For more than a decade there has been great confusion over the semantics of the standard "arithmetic right shift" instruction. This confusion particularly afflicts authors of computer reference handbooks and of optimizing compilers. The fact that shifting is not always equivalent to division has been rediscovered over and over again over the years, but has never been publicized.

This paper quotes a large number of sources to prove the widespread extent of this confusion, and then proceeds to a short discussion of the problem itself and what to do about it.

Keywords: arithmetic shift, twos complement, ones complement, division, remainder, binary arithmetic

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

* NSF Fellow

Recently I was looking through some files concerning BLISS which Bill Wulf had given me access to and came across a letter from Bruce Leverett to him concerning a bug:

> Bill -
>
> Scandal! Consider an expression like A←.A/2 . The code we put out for this expression is exactly one instruction:
>
> ASR A
>
> But that ain't right. Suppose A is negative, for instance -1. Dividing -1 by 2 gives a quotient of 0 and a remainder of -1 (See the DIVMOD routine if you want to check this out). But shifting -1 right by one bit gives -1. [1]

I was amused at first, but then recalled that the MacLISP compiler had had exactly this bug last year - it incorrectly compiled divisions by powers of two into arithmetic right shifts. When we (Jon L. White and I) found the bug, we were properly horrified, but then simply changed the compiler not to perform that particular "optimization" and shrugged it off.

Having now read of the similar experience of the BLISS team, I began to wonder how many other optimizing compilers in the world contain this same error? I was at home on a Friday night at this time, but for fun I started thumbing through what books and old manuals I happened to have on my shelf. The results were appalling. With only an hour's research I found more than a dozen cases of this same confusion! Let me give some examples here. (Unless otherwise stated, all machines mentioned here use twos complement arithmetic, which is the crux of the matter, as I shall explain below.)

In 1968 DEC's PDP-10 System Reference Manual stated:

> A single shift left is equivalent to multiplying the number by 2 (provided no bit of significance is shifted out); a shift right divides the number by 2... A positive E produces motion to the left, a negative E to the right; -E is thus the power of 2 by which the number is multiplied. [2]

Also in 1968, the handbook for the DRC-44 contained this instruction description:

> Shift the 23 bit A register right N times... The result is division by 2^N. [3]

In 1969 DEC's PDP-11 Handbook stated:

> ASR performs a signed division of the destination by two. [4]

In 1970 the DEC Small Computer Handbook began a description of a right shift this way:

> This instruction is used for scaling... [5]

General Automation's 1972 publication The Value of Power stated:

> A right shift is more useful, because it is equivalent to dividing by 2, while the left shift is equivalent to multiplying by 2 and can be reproduced by adding the contents of a register to itself. [6]

Not only reference handbooks are to blame. An IBM tutorial text of 1966 on the System/360 contains the following discussion:

> After rounding off we are left with eleven superfluous bits at the right end of the product. These can be shifted off the end of the register with a suitable shift instruction. "Suitable" in this case means that the shift should be to the right, ... and it should be an algebraic shift so that if the number were negative, proper sign bits would be shifted into the register... The point of doing all this is that we have replaced a Divide with a Shift, and the latter is considerably faster than the former.

Ironically, the next page states:

> We present examples like these to warn the unwary and to lay a foundation of understanding for those with problems where the advantages of binary arithmetic are worth the care that must be exercised in using it. [7]

The confusion of shifting and division is not confined to manufacturers' publications. Gries' 1971 book Compiler Construction for Digital Computers

asserts:

> On binary machines, integer multiplication or division by a power of two can be performed by a shift. [8]

The 1975 InterLISP Reference Manual provides a totally erroneous definition of its right shift function:

> (arithmetic) right shift, value is n*2^-m, i.e. n is shifted right m places. n can be positive or negative. [9]

To be sure, later the user is given a warning in the form of a single example:

> Note that shifting (arithmetic) a negative number "all the way" to the right yields -1, not 0.

but this occurs much later in the page. Furthermore, the on-line InterLISP documentation system, helpsys, prints out only the erroneous definition and not the caveat.

Even ANSI is not clear on the shifting problem. Their 1970 definition of arithmetic shift is as follows:

> arithmetic shift. (1) A shift that does not affect the sign position. (2) A shift that is equivalent to the multiplication of a number by a positive or negative integral power of the radix. [10]

This definition ignores all questions of overflow, and does not recognize that the two definitions are not compatible on a machine with no minus zero if division implies truncation towards zero.

At this point let us consider the problem of shifting more closely. Why is not a shift equivalent to a division? In 1968 the IBM System/360 Principles of Operation manual gave the following exposition:

> A right shift of one bit position is equivalent to division by 2 with rounding downward. When an even number is shifted right one position, the value of the field is that obtained by dividing the value by 2. When an odd number is shifted right one position,

> the value of the field is that obtained by dividing the next lower number by 2. For example, +5 shifted right by one bit position yields +2, whereas -5 yields -3. [11] [emphasis in the text]

This at least gives the reader some warning that shifting and division are not the same thing; but it is too easy to read "toward zero" after the word "downward", and furthermore the rule given supplies no insight as to the action of a multiple-position shift on a negative number.

A 1972 Data General Corporation publication sheds a little more light on the matter:

> If ones complements were used for negatives, one could read a negative number by attaching significance to the 0s instead of the 1s. In twos complement notation... one can read a negative number by attaching significance to the rightmost 1 and attaching significance to the 0s to the left of it... In a negative (proper) fraction, 0s may be discarded at the right; as long as only 0s are discarded, the number remains in twos complement form because it still has a 1 that possesses significance; but if a portion including the rightmost 1 is discarded, the remaining part of the fraction is now a ones complement. Truncation of a negative integer thus increases its absolute value.

Unfortunately this lucid discussion is followed on the next page by the simple assertion:

> Shifting one place to the right divides by 2. Truncation occurs at the right, and a bit equal to the sign must be entered at the left. [12]

Unless the reader were exceptionally careful, he could easily take "truncation" in the FORTRAN sense, i.e. "rounding toward zero".

Knuth also provides a clear exposition of the problem, though in the context of decimal and not binary arithmetic:

> The major difference between signed magnitude and ten's complement notations in practice is that shifting right does not divide the magnitude by ten; for example, the number -11=...99989, shifted right one, gives ...99998=-2 (assuming that

> a shift to the right inserts "9" as the leading digit when the number shifted is negative). In general, x shifted right one digit in ten's complement notation will give $\lfloor x/10 \rfloor$, whether x is positive or negative. [13]

Knuth does fail to mention explicitly that shifting is equivalent to division in nines (i.e. ones) complement notation. This fact is not difficult to see if one considers that in ones complement notation 0 and 1 bits play roles which are precisely dual; thus if shifting is equivalent to division (with rounding toward zero!) for positive numbers, a fact which is not difficult to prove, then by duality it is equivalent for negative numbers.

Another way of viewing the problem is to recognize that there are two reasonable definitions of defining integer division. In any case, given a dividend n and divisor d, we wish to compute a quotient q and remainder r such that $qd+r=n$ and $|r|<|d|$. One definition of the division operation, which I call "FORTRAN-style", is to require that r be zero or of the same sign as n. This results in a symmetry around zero: if FORTRANDIVIDE(n,d) produces (q,r), then FORTRANDIVIDE(-n,d) produces (-q,-r). Put another way, $q=\text{sign}(n)\lfloor|n/d|\rfloor$. Another definition, more elegant perhaps to number theorists, I call "modulus-style". In fact there are two ways to define modulus-style division: one may either require that r be non-negative, or that r have the same sign as d. (In these cases $q=\text{sign}(d)\lfloor n/|d|\rfloor$ and $q=\lfloor n/d\rfloor$.) (The author is indebted to Henry G. Baker for pointing this cases out.) In the case of a shift, the effective divisor is always positive, and so the two definitions are equivalent. (One can imagine yet other definitions of division subject to the restriction that $|r|<|d|$; for example, "balanced" division, in which r lies between d/2 (inclusive) and -d/2 (exclusive); for d=3, this yields a "balanced ternary" remainder. In general, for given d there are $2|d|-1$ ways to define division satisfying $|r|<|d|$.)

On a ones complement machine, arithmetic right shift performs a FORTRAN-style division; the bits shifted out, when preceded by the sign bit of the operand, may be interpreted as the remainder. On a twos complement machine, arithmetic right shift performs a modulus-style division; the bits shifted out may be interpreted as the (non-negative) remainder. On either kind of machine, however, the division instruction usually implements the FORTRAN-style definition, no doubt primarily due to the influence of FORTRAN itself. Later high-level languages have in turn tended to adopt the FORTRAN-style definition of integer division.

Some handbooks for ones complement computers in fact correctly describe arithmetic right shifts as being equivalent to (FORTRAN-style) division [14] [15]; so also did handbooks for sign-magnitude machines. [16] I suspect that this early influence is what leads many documentors and users of more modern twos complement machines astray.

What I find distressing, however, is not so much that people keep making this same mistake about shifting on twos complement computers, as that they do not seem to publicize this mistake, even within their own group or company. There is evidence that the IBM System/360 FORTRAN IV (H) had the same bug in 1966 that a decade later plagued the MacLISP and BLISS compilers. The program logic manual of that time stated:

> Multiplication and division by powers of two are converted, respectively, to left and right shift operations. [17]

Furthermore, in a 1969 CACM article (last revised in 1968) on FORTRAN IV (H), Lowry and Medlock said:

> If one operand is a power of two, a number of improvements may be possible: Integer multiplication or division may become a shift... [18]

The bug was eventually fixed, for the 1968 edition of the program logic manual merely stated:

> Multiplications by powers of two are converted to left shift operations. [19]

Despite this horrendous error in the compiler, however, the 1969 edition of the IBM tutorial text cited above still contained the same error it did in 1966! [20]

A similar example occurs in DEC's PDP-10 Reference Handbook. The editions of 1971 and later contain the same text as the 1968 manual cited earlier, but in addition contain a note in the margin:

> An arithmetic right shift truncates a negative result differently from IDIV if 1s are shifted out. The result of the shift is more negative by one than the quotient of IDIV. To obtain the same quotient that IDIV would give with a dividend in A divided by N=2^K, use
>
> ```
> SKIPGE A
> ADDI A,N-1
> ASH A,-K
> ```
>
> This takes 5-6 us as opposed to about 16 us for IDIVI. [21]

This idea is worth commenting on; it is the only suggestion I found for "fixing" the problem. One can think of it as a variation on the old FORTRAN trick for rounding numbers by adding 0.5 and then truncating; adding N-1 to the negative number effectively turns the floor operation of the ASH instruction into a ceiling operation. A more interesting way to view it is in terms of the following code:

```
SKIPGE A
SUBI A,1
ASH A,-K
SKIPGE A
```

```
ADDI A,1
```

Recalling that a twos complement negative is formed by taking the ones complement and adding 1, we can see that this piece of code essentially converts the operand to ones complement (skip if the operand is positive, else subtract 1), performs the arithmetic shift (which performs FORTRAN-style division on ones complement numbers of either sign), and then converts back to twos complement. It is not difficult to prove that the shift and the addition can be permuted if the added quantity is adjusted by the shift factor:

```
SKIPGE A
SUBI A,1
SKIPGE A
ADDI A,N
ASH A,-K
```

Combining the addition and the subtraction yields the code given in the PDP-10 handbook. (It also removes a fencepost error which occurs when the number to be shifted is the most negative number.)

To return, however, to my earlier complaint: besides fixing the PDP-10 handbook, DEC also removed the comment about scaling from the Small Computer Handbook in 1971. [22] However, the PDP-11 handbooks were still in error in 1973 and 1975! [23] [24]

The lesson is clear: not only is this mistake made over and over and over again, but we have not learned from it. I think that, to those of us that fall prey to it, when we finally discover it for ourselves it seems to be such

a trivial error that we do not mention it to others, either because it seems not worth mentioning or because we do not wish to appear foolish for making such a "simple" mistake. I have written this paper partly to show all such people that they are not alone. I also have three exhortations to make:

(1) To documentors: whenever documenting a standard arithmetic shift instruction for a twos complement machine, write (in bold face!) that it is not the same as a FORTRAN-style divide instruction. Give a complete discussion of this fact, and mention Knuth's floor function description and the "fix" given in the PDP-10 Reference Handbook, or other appropriate ways to fix the problem.

(2) To computer manufacturers: arithmetic shifts actually seem to be pretty useless. Left shifts are usually the same as logical left shifts (except for overflow detection); right shifts don't do what the user wants. Why not implement the PDP-10's solution in hardware as a single shift instruction? This can also be done by incrementing the result of an arithmetic right shift iff the operand was negative and a 1 bit was shifted out. On the other hand, the FORTRAN-style division is probably never used for remainder with a negative dividend anyway; it might be more convenient all around to implement modulus-style division and leave shifting alone. Several number theorists have indicated to me that FORTRAN-style division is useless for their purposes, and furthermore it is easier to simulate FORTRAN-style given modulus-style than vice versa.

(3) To compiler writers and maintainers: check your compiler now for this bug! And in the future, be very careful about installing new "optimizations". If possible, try to prove that they work. After all, a bug in the compiler

will produce bugs in many programs.

To anyone who by now still thinks that the problem is "obvious" and "trivial", I suggest that he think about arithmetic left shift on a ones complement machine: it does not shift in zeros from the right, but copies of the sign bit. "There's glory for you!" [25]

In conclusion, let me quote the rest of the letter from Leverett to Wulf that started me thinking about all this. The last sentence is quite chilling.

> It's clear what the fix for this is in the compiler--just change DMULDIVMOD [the routine responsible for optimizing multiply, divide, and modulus operators - GLS] so that it doesn't try to change divisions into shifts. The hairy problem is compatibility. To give you an idea of how many programs are out there which will not work when we fix the compiler so that it's right, consider MULDIV.B11. Several times in the multiply and divide routines we shift a number one bit to the right, by telling the compiler to divide it by 2. Of course this is wrong source code. And of course fixing it is easy--just change all the divides into shifts (which they should have been in the first place). But what's the best way to tell people that they have to comb through their source files looking for this error, quick before they get the next version of the compiler?
>
> The first thing I'd like to do is send a note to all the Bliss-11 users in the department telling them what's wrong. The way to send a note to all the Bliss-11 users, I would guess, is to figure out which projects are doing coding in Bliss-11, and send mail to project numbers (e.g. N810). Do you know offhand which projects are making use of Bliss-11?
>
> I'm not so worried about non-CMU users. When I send one of them a new compiler, it's because that user specifically asked for one, and he always has an old one to fall back on. If I make sure to send him reams of many-times-redundant documentation about the difference between the old and new compilers, I have done as much as I can to help him with his compatibility problems. It's the local users that bother me. In particular, there's this operating system kernel...
>
> Bruce

References

[1] Personal communication, Bruce Leverett to William A. Wulf. (Spring 1976) Quoted by permission of Mr. Leverett.

[2] Digital Equipment Corporation. PDP-10 System Reference Manual. Order No. DEC-10-HGAA-D (May 1968), p. 2-31.

[3] Dynamics Research Corporation. Instruction List for DRC Model 44 General Purpose Processor. H-18A (September 1968), 34.

[4] Digital Equipment Corporation. PDP-11 Handbook. (1969), 35.

[5] Digital Equipment Corporation. Small Computer Handbook. (1970), 60.

[6] General Automation. The Value of Power. Document Number 89A00064A-A (1972), p. 5-19.

[7] IBM Corporation. A Programmer's Introduction to the IBM System/360 Architecture, Instructions, and Assembler Language. Form C20-1646-1 (May 1966), 62-63.

[8] Gries, David. Compiler Construction for Digital Computers. John Wiley and Sons (New York, 1971), 411.

[9] Teitelman, Warren, et al. InterLISP Reference Manual. Xerox Corporation (Palo Alto, December 1975), 13.4.

[10] American National Standards Institute. ANS Vocabulary for Information Processing. (1970) Reprinted by IBM Corporation in Data Processing Glossary, Form GC20-1699-3 (June 1972).

[11] IBM Corporation. IBM System/360 Principles of Operation. Form A22-6821-7 (September 1968), 33.

[12] Data General Corporation. How to Use the Nova Computers. (October 1972), p. 2-10.

[13] Knuth, Donald E. The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley (Reading, Mass., 1969), 169-170.

[14] Digital Equipment Corporation. Programmed Data Processor-1 Handbook. (1963), 18.

[15] Digital Equipment Corporation. Laboratory Computer Handbook. (1971), 154.

[16] IBM Corporation. Reference Manual 709-7090 Data Processing System. Form A22-6503-1 (November 1959).

[17] IBM Corporation. IBM System/360 Operating System FORTRAN IV (H) Program Logic Manual. Form Y20-0012-0 (1966), 26.

[18] Lowry, Edward S., and Medlock, C.W. Object Code Optimization. Comm. ACM 12, 1 (January 1969), 21.

[19] IBM Corporation. IBM System/360 Operating System FORTRAN IV (H) Compiler Program Logic Manual. Form Y28-6642-3 (November 1968), 26.

[20] IBM Corporation. A Programmer's Introduction to IBM System/360 Assembler Language. Form C20-1646-5 (July 1969), 35-36.

[21] Digital Equipment Corporation. PDP-10 Reference Handbook. Order code ATX (1971), p. 1-49.

[22] Digital Equipment Corporation. PDP-8/e Small Computer Handbook. (1971), p. 7-4.

[23] Digital Equipment Corporation. PDP-11/45 Processor Handbook. (1973), 53.

[24] Digital Equipment Corporation. LSI-11 PDP-11/03 Processor Handbook. (1975), p. 4-13.

[25] Dumpty, Humpty. Quoted in Carroll, Lewis, Through the Looking Glass, Chapter VI.